BLACK CODHER

CODING PROGRAMME

# Black Codher Bootcamp

BLACK CODHER

CODING PROGRAMME

# UNIT 4 - Session 6
# React & React Native

# End of Session 5 Summary

1. Reviewing Bookcase App Homework
2. Nested components
3. Using Proptypes for better validation and type checking
4. Lifting State Up
5. Event Handling
6. Continuing with the Bookcase App

## Goals for Unit 4 - Session 6

1. React routes
2. React web pages
3. Fragments
4. Form Management
5. Controlled Components
6. Continuing with the Bookcase App

BLACK CODHER

CODING PROGRAMME

## React Routes

# React Routes

- Routing is the ability to move between different parts of an application when a user enters a URL or clicks an element within the application (e.g. link, tab, button etc...)

- To add routes to the React application we can use a popular routing library called **react-router-dom**

- The **react-router-dom** is designed to be used for web applications

- To use the **react-router-dom** we will need to install the library using a package manager

```
> npm install react-router-dom
```

## React Routes

BLACK CODHER

- There are a few different types of routers. **HashRouter**, **MemoryRouter** and **BrowserRouter**.

- The **HashRouter** uses the hash portion of the URL (the window.location.hash) to keep the UI in sync with the URL (e.g. <a href="**#/bookcase/first**">)

- The **MemoryRouter** keeps the history of the URL in memory, so it doesn't read or write to the address bar

- Our React bookcase app is browser based so we will use the **BrowserRouter**.

- The **BrowserRouter** uses the HTML5 history API (pushState, replaceState and the popstate event) to keep your UI in sync with the URL

# React Routes

- To use React routes in a in a React application we will need to import the BrowserRouter at the top of the file

```
import { BrowserRouter as Router, Route } from 'react-router-dom'
```

- In the import statement above we are importing the **BrowserRouter** and the Route components. We are also aliasing **BrowserRouter** as Router. This will mean we can refer to the component as Router when used in our functional component

- Aliasing is used when we find ourselves requiring imports from multiple different modules that contain exports named the same or if you want to use a shorter reference name in your code

## React Routes

- The **<Route/>** component is the base building block in the React Router library. It renders the appropriate user interface when the current location matches the route's path

- It is also important to note that any Router expects one child component. React.Fragments* can be used to overcome this issue

- The Router component provides three props that can be used to determine which component to render:

1. component prop
2. render prop
3. children prop

## React Routes

BLACK CODHER

### Component Prop

- The component prop defines the React element that will be returned by the Route when the path is matched:

```
<Router>
  <Route exact path="/" component={libraryItems} />
</Router>
```

- In the example above the components 'libraryItems' in the **component** prop will be returned when the browser url matches the path attribute. In the example above the path is set to home directory

# React Routes

## Render Prop

- The render prop provides the ability for inline rendering and passing extra props to the element

- The render props expects a function that returns a React element when the browser URL matches the current path attribute:

```
<Router>
  <Route exact path="/bookcase" render={() => (
    <BookList books={books} addToBookcase={addToBookcase}/>)} />
</Router>
```

## React Routes

### Children Prop

- The children prop is similar to the render prop. It always expects a function that returns a React element.

- However with the **children** prop the element is returned for all paths regardless of current location

```
<Route children={props => <Footer {...props} /> } />
```

## React Routes

### Router Link Component

- The `<Link/>` component is used to navigate to the different parts of an application by way of hyperlinks.

- It's similar to HTML's anchor element (`<a href="/">link</a>`) but the main difference is that using the Link component does not reload the page, it renders the elements based on the matching Router

- Using an anchor tag would require that the page is reloaded in order to load the new user interface. When the Link component is clicked, it re-renders the page and also updates the URL

## React Routes

Link syntax:

```
<Link to="/items">Items</Link>
```

- The <Link/> component uses a **to** prop to define the location to navigate to. This prop can either be a string or a location object

## Fragments

## Fragments

- A component can return only one base element however a common pattern in React is for a component to return multiple elements

```
render(){
  return(
    <div>First</div>
    <div>Second</div> //This will throw a SyntaxError
  );
}
```

- React.Fragment and the new, shorter syntax can solve this problem

# Fragments

- Fragment provides a base element that can be used to group a list of children without adding extra nodes to the DOM, reducing bloat

```
render(){
  return(
    <React.Fragment>
      <div>First</div>
      <div>Second</div>
    </React.Fragment>
  );
}
```

- Fragments declared with the explicit <React.Fragment> syntax may have keys. Key is the only attribute that can be passed to a Fragment

## Fragments

- Keys are required when a child element in a list requires unique "keys". Without the key attribute React will raise a key warning.

- The new short syntax will allow you to use what looks like empty tags <></> instead of React.Fragement:

```
render(){
  return(
    <>
      <div>First</div>
      <div>Second</div>
    </>
  );
}
```

- No attributes can be passed using the short syntax

## Exercise 1

# Exercise 1: Adding Routes

- We will be adding two **BookList** components to our **mybookcase** application. The first **BookList** will return the list of books in our book.json file. The second **BookList** will be a list of books we are "saving" for later.

- In order to navigate between our **BookLists** we will need to use Routes (**react-router-dom** library). We will add a **Header** component which we can reuse in each of our routes. This will contain our navigational links

1. Open the **mybookcase** app. Add a file called **Header.js** to the components folder

2. In **Header.js** create a new functional component called **Header**

## Exercise 1: Adding Routes

3. In the **Header.js** file add an import statement to the top of the file importing the **Link** element from the **react-router-dom** library

```
import {Link} from 'react-router-dom';
```

4. In the return block of the **Header** component return:
   - A header title e.g. `<h1>My Bookcase</h1>`
   - Two `<Link>` components

These components will be displayed at the top of each page:

```
    <Link to="/"> Home </Link>
    <Link to="/bookcase" className="bookLink"> Bookcase</Link>
```

This will mean navigating to localhost:3000 or localhost:3000/bookcase when our app is running will take us to our new views

# Exercise 1: Adding Routes

5. Open the **App.js** file and import the **BrowserRouter** and the **Route** components from the **react-router-dom** library

```
import { BrowserRouter, Route } from 'react-router-dom'
```

6. Import the **Header.js** from the **App.js** file

```
import Header from './components/Header';
```

7. Add one **BrowserRouter** component with two **Route** components. The Route components should have **path** attributes matching the **to** attributes of the Link components in the **Header.js** file

8. Add the Header component to both routes e.g.

# Exercise 1: Adding Routes

Example home page route in App.js:

```
<BrowserRoute>
  <Route exact path="/" render={() => (
    <>
      <Header />
      <BookList books={books} addBook={addBook} />
    </>
   )} />
  //Add other route here
</BrowserRoute>
```

7. Add our **BookList** component to both the home page (path="/") and the bookcase (path="/bookcase") routes

- Each **BrowserRouter** can be thought of as a view. When a link in the Header is clicked a new 'view' will be rendered

## Checkpoint!

**How are you feeling?**

RED - I have no idea what you're talking about.

YELLOW - I have some questions but feel like I understand some things.

GREEN - I feel comfortable with everything you've said.

BLACK
CODHER
CODING PROGRAMME

## React Web Page

## React Web Pages

- The **react-router-dom** allows us to create a multiple page website so when a user enters a URL in the address bar the browser navigates to a corresponding page
- Each new page is still a component that returns JSX markup
- To add an about us 'page' we can add a new component called AboutUsPage
- The AboutUsPage component will return a component with the relevant page content
- To add a link to the AboutUsPage component we would need to add a <Link> imported from the **react-router-dom** library

## React Web Pages

```
const AboutUsPage = (props) => {
     return (
         <div className="header">
             <h1>About</h1>
             <div className="breadcrumb">
                 <Link to="/"> Home </Link>
             </div>
             <div className="page">
           <h1>Welcome to the Bookcase Application</h1>
                <p>This React application allows a user to add a set of
 books to a collection of books. It connects to the Google Books API
 and...</p>
        </div>
         </div>
     )
```

BLACK CODHER

CODING PROGRAMME

## Form Management

# Forms & Controlled Components

- HTML forms work differently from other DOM elements in React, because form elements naturally keep some internal state

- Standard HTML form data is usually handled by the DOM. In React form data is handled by components. Example of a default HTML form syntax

```
<form>
  <div>
      <label for="username">Username:</label>
      <input type="text" id="username" name="username">
  </div>

  <div>
      <label for="pass">Password:</label>
      <input type="password" id="pass" name="password"/>
  </div>

  <input type="submit" value="Sign in">
</form>
```

## Forms & Controlled Components

- Forms have default HTML form behavior of browsing to a new page when the user submits the form. If you want this behavior in React, it just works
- However, to implement a Javascript function that handles the form submission we would use a technique called "**controlled components**"
- In HTML form elements such as **input**, **textarea** and **select** maintain their own state and update based on user input
- In React, state is kept in the state property and updated with the **setState** function or **useState** Hook
- The two methods for keeping state can be combined to make the React state be the "single source of truth"

## Forms & Controlled Components

- An example of a functional component that returns a form with controlled components:

```
const LogForm = () => {
const [name,setName] = useState('');

function handleSubmit(event) {
  event.preventDefault();
  alert('A name was submitted: ' + name);
}
  return (
    <form onSubmit={(e) => handleSubmit(e)}>
       <label>
         Name:
         <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
       </label>
       <input type="submit" value="Submit" />
     </form>
   );
}
```

## Forms & Controlled Components

- Since the **value** attribute (line 12) is set on our form element, the displayed value will always be name, making the React state the source of truth
- Since the **onChange** function of the **input** field runs on every keystroke to update the React state, the displayed value will update as the user types
- With a **controlled component**, the input's value is always driven by the React state
- While this means you have to type a bit more code, you can now pass the value to other UI elements too, or reset it from other event handlers

BLACK CODHER

CODING PROGRAMME

## Exercise 2

## Exercise 2: Building a Form

1. Get the latest files from the **black-codher-bootcamp** by navigating to the repository on your local system and pulling the latest:

```
> cd black-codher-bootcamp
> git pull
```

2. Open folder **black-codher-bootcamp\unit04-react\session6**. Copy the **index.js** into your existing **test-project**.

3. To test the file navigate to your **test-project** folder and run npm start:

```
> cd test-project
> npm start
```

## Exercise 2: Building a Form

4. The form contains a name field which is a **controlled component**. Add the following form fields as controlled components:

```
<label>Age <input type="number"/></label>
<label>Location <input type="text"/></label>
```

5. Ensure the state (value) of the elements are updated when the form is updated

6. Display the submitted values on the page instead of an alert. Add a <div></div> after the closing </form> to display the data.

7. Use a React.Fragment or short syntax (<></>) to encapsulate the form and <div> elements

## Exercise 2: Building a Form

8. Add a message "Thank you for submitting your form" to the output displayed in the <div></div> whenever the form is submitted

9. Ensure the message "Thank you for submitting your form" is removed whenever the user changes an input field.  Hint: add a React Hook that sets a variable when the form is submitted, e.g:

```
const [hasSumitted, setSubmitted] = useState(false);
```

# Checkpoint!

BLACK CODHER

**How are you feeling?**

RED - I have no idea what you're talking about.

YELLOW - I have some questions but feel like I understand some things.

GREEN - I feel comfortable with everything you've said.

# Session 6 Summary

1. React routes
2. React web pages
3. Fragments
4. Form Management
5. Controlled Components
6. Continuing with the bookcase app

## Homework

## Homework - Add About Page

BLACK CODHER

- Add an About page to your mybookcase app. Ensure you can navigate to it from any page/view on the application

1. Open the **mybookcase** app

2. Create a new folder in the **src** folder called **pages**

```
> cd .\mybookcase\src\
> mkdir pages
> cd pages
```

3. In the pages folder create a new file called **About.js**

```
> echo . > About.js
```

# Homework

BLACK CODHER

4. Open **About.js** in Visual Code and add add a new functional component called About

5. Add a header and a description of the functionality of the app e.g.

**Welcome to the Bookcase Application**

The following application was created by <insert your name>.
This bookcase app displays a list of books that a user can save to a local bookcase

Click on the "Add + " button to add a book to your bookcase.
Use the search bar to find the latest books by name, author or description

6. Style the About page by creating an external css file e.g. About.css

## Homework

7. Add a <Link> to the About page that will return the user back to the home page e.g. `<Link to="/">Home</Link>`

8. Add the link to the **Header.js** file to ensure it is displayed on each view